

# MethodSCRIPT SDK Example - C

---



Last revision: March 22 2019

© 2019 PalmSens BV

[www.palmsens.com](http://www.palmsens.com)

## 1.1 Contents:

The example *MethodSCRIPTExample.c* found in the */MethodSCRIPTExample-C* folder demonstrates basic communication with the EmStat Pico using the SDK (C libraries). The example allows the user to start measurements on the EmStat Pico from a Windows PC using a simple C program which makes use of the MethodSCRIPT SDK (C libraries).

### 1.1.1 Basic Console Example (MethodSCRIPTExample.c)

This example demonstrates how to implement serial communication with the EmStat Pico to

- Establish a connection with the device
- Write the MethodSCRIPT that defines the measurement's parameters
- Read and parse the measurement response from the device

This does not include error handling, method validation etc.

## 1.2 Communications

The MSComm.c from the MethodSCRIPT SDK (C libraries) acts as the communication object to read/write from/to the EmStat Pico.

The functions in the code snippet below are the necessary read/write functions defined in the C example. WriteToDevice writes a char to the EmStat Pico and ReadFromDevice returns the first byte from the read buffer as soon as there is data available in the buffer.

```
int WriteToDevice(char c)
{
    char writeChar[2] = {c, '\0'};
    if (WriteFile(hCom, writeChar, 1, &dwBytesWritten, NULL))
    {
        return 1;
    }
    return 0;
}

int ReadFromDevice()
{
    char tempChar; //Temporary character used for reading
    DWORD noBytesRead;
    ReadFile(hCom, //Handle of the Serial port
             &tempChar, //Temporary character
             sizeof(tempChar), //Size of TempChar
             &noBytesRead, //Number of bytes read
             NULL);
    return (int)tempChar;
}
```

The read/write functions are required to initiate the MSComm communication library.

```
MSComm msComm;
RetCode code = MSCommInit(&msComm, &WriteToDevice, &ReadFromDevice);
```

### 1.2.1 Connecting to the device

The following code snippet shows how to open a serial com port using the Windows API. In order to use the Windows API for serial communication, the windows.h header has to be included in the C program's header file as below.

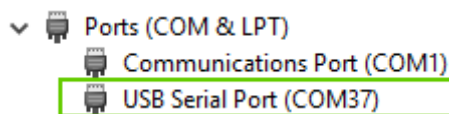
```
#include <windows.h>
```

A valid handle to the port to which the EmStat Pico is connected is necessary for the read/write functions in the MSComm library. The code below can be used to open the com port connected to the device.

```
HANDLE hCom;
hCom = CreateFile(PORT_NAME, GENERIC_READ | GENERIC_WRITE, 0, // must be
opened with exclusive-access
    NULL, // no security attributes
    OPEN_EXISTING, // must use OPEN_EXISTING
    0, // not overlapped I/O
    NULL // hTemplate must be NULL for comm devices
);

const char* PORT_NAME = "\\.\COM37";
```

The name of the com port connected to the device can be found in the Device Manager in Control Panel in Windows as shown below.



Once a valid handle is created, the baud rate (230400) for the device has to be set using the device control block object.

## 1.2.2 Sending the MethodSCRIPT

The measurement configuration parameters can be read from a MethodSCRIPT file stored on the PC. In this example, the MethodSCRIPT files are stored in the ScriptFiles directory. The code snippet below is used in the example to read the MethodSCRIPT from the file and in turn write it to the device.

```
int SendScriptFile(char* fileName)
{
    FILE *fp;
    char str[100];

    fp = fopen(fileName, "r");
    if (fp == NULL) {
        printf("Could not open file %s", fileName);
        return 1;
    }
    while (fgets(str, 100, fp) != NULL) // Reads a single line from
the script file and writes it on the device.
    {
        WriteStr(&msComm, str);
    }
    fclose(fp);
    return 1;
}
```

## 1.2.3 Receiving response

This example uses the MSComm library to receive and parse the data returned by a measurement. In order to read and parse the response from the device, the Receive Package function from the MSComm library can be used. This function requires a reference to an initiated MSComm struct (msComm) and it returns the parsed data in the referenced MeasureData struct (data)

```
code = ReceivePackage(&msComm, &data);
```

### 1.2.4 Parsing the response

Each line of response returned by the function `ReadBuf()` in `MSComm` library, can be further parsed if it is identified to be a data package. Here's a sample response (raw data) from a Linear sweep voltammetric measurement.

```
eM0000\n
Pda7F85F3Fu;ba48D503Dp,10,288\n
Pda7F9234Bu;ba4E2C324p,10,288\n
Pda806EC24u;baAE16C6Dp,10,288\n
Pda807B031u;baB360495p,10,288\n
*\n
\n
```

While parsing the response, various identifiers are used to identify the type of response packages. For example, In the above sample response package,

1. 'e' marks the beginning of a response.
2. 'M' marks the beginning of a measurement loop.
3. 'P' marks the beginning of a row of data package.
4. "\*/\n" marks the end of a measurement loop.
5. "\n" marks the end of response.

The following information can be found in the data packages received from the device.

- Potential (set cell potential in V)
- Current (measured current in A)

In case of Impedance spectroscopy measurements, the following data values can be obtained from the response.

- Frequency (set frequency in Hz)
- Real part of complex Impedance (measured impedance Ohm)
- Imaginary part of complex Impedance (measured impedance in Ohm)

The following meta data values if present can also be obtained from the data packages.

- CurrentStatus (OK, underload, overload, overload warning)
- CurrentRange (the current range in use at the moment)
- Noise (Noise)

#### 1.2.4.1 Parsing the parameter values

Each row of data package begins with the header 'P'. The parameters from the data package line can be then split in to tokens based on the delimiter ';'. Each of the parameters separated from the package line can be then parsed to get the actual values of the parameters.

The potential readings are identified by the string *da*

The current readings are identified by the string *ba*

The frequency readings are identified by the string *dc*

The real impedance readings are identified by the string *cc*

The imaginary impedance readings are identified by the string *cd*

For example, in the sample package seen above, the parameter identifiers are  
*da7F85F3Fu* - *da* Potential reading and  
*ba48D503Dp,10,288* - *ba* current reading.

The following 8 characters hold the parameter value. The parameter value for current reading (8 characters) from the above sample package is *48D503Dp*. This value is further parsed to retrieve the actual parameter value with the respective unit prefix.

The SI unit prefix from the package can be obtained from the parameter value at position 8

In the above sample package, the unit prefix for current data is 'p' which is 1e-12 A.

After obtaining the parameter identifier and the parameter values from the package, the meta data values if present can be parsed. Meta data values if present are separated by the demiliter character ','.

## 1.2.4.2 Parsing the meta data values

The meta data values are separated based on the delimiter ',' and each of the values is further parsed to get the actual value.

The first character of each meta data value `metaData[0]` identifies the type of meta data.

'1' – status

'2' – Current range index

'4' - Noise

The status is 1 character hex bit mask. It is converted to long int. The status can be obtained as shown in the code snippet below.

For example, in the above sample, the available meta data values for current data are, 10,288. The first meta data value is 10.

1 – meta data status – 0 indicates OK.

The meta data type current range is 2 characters long hex value. If the first bit high (0x80), it indicates a high speed mode current range.

The code below can be used to get current range bits from the package.

The hex value is then converted to int to get the current range string as shown below.

For example, in the above sample, the second meta data available is 288.

2 – indicates the type – current range

88 – indicates the hex value for current range index – 1mA. The first bit 8 implies that it is high speed mode current range.